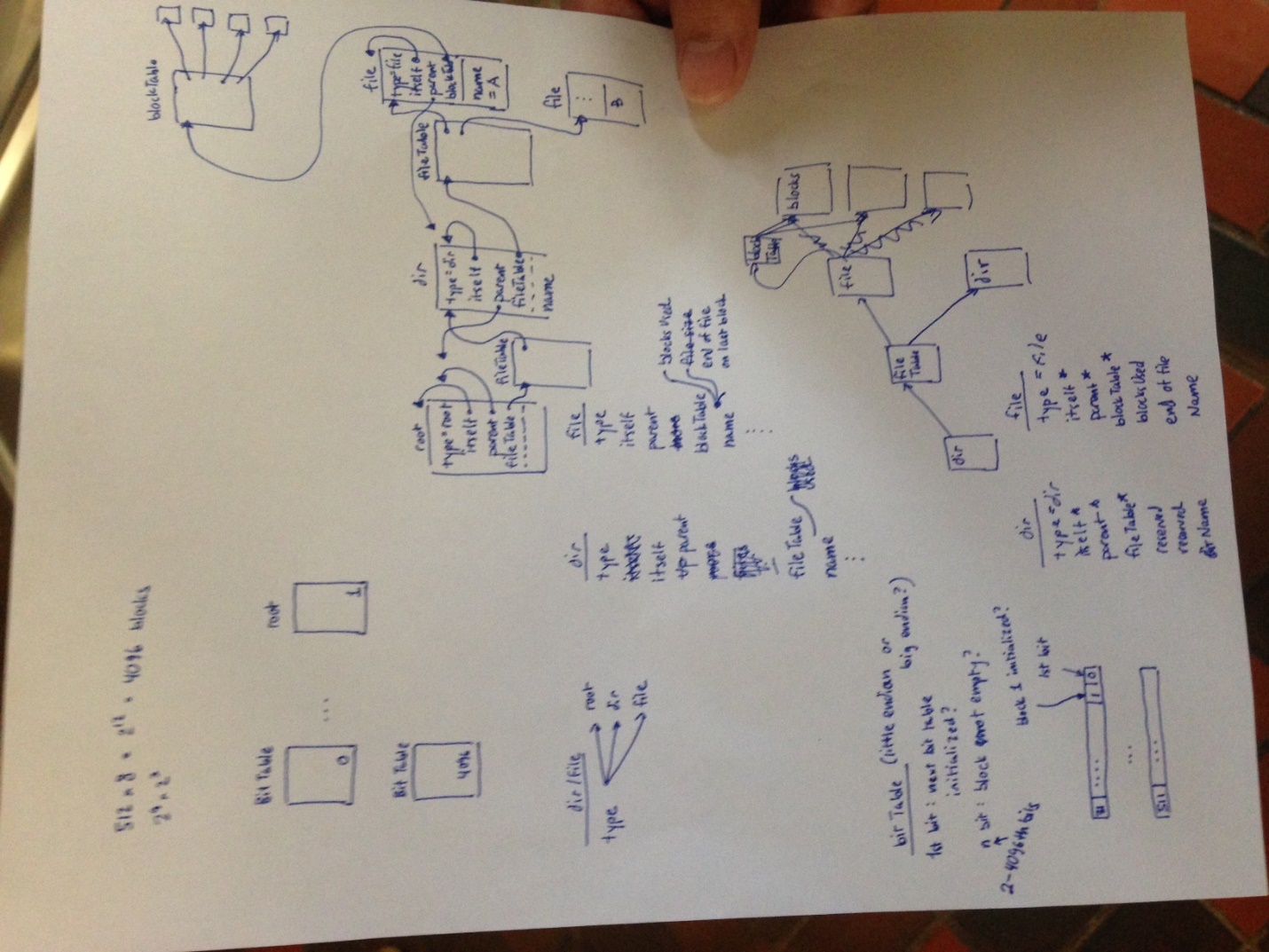
1. Objectives

In this lab we designed and implemented a persistent filesystem that interfaced the launchpad with an SD card. We also implemented interpreter commands that allow us to send commands directly to the middle-level filesystem interface to make, write to, and read from files.

1. Hardware Design
2. Software Design

(see efile and disk; image of design attached)



E) Analysis and Discussion (2 page maximum). In particular, answer these questions

1) Does your implementation have external fragmentation? Explain with a one sentence answer.

Yes; file blocks are allocated as needed, so they will become spread out across the disk over time.

2) If your disk has ten files, and the number of bytes in each file is a random number, what is the expected amount of wasted storage due to internal fragmentation? Explain with a one sentence answer.

(sblk - 9) / 2, sblk = size of disk in blocks

3) Assume you replaced the flash memory in the SD card with a high speed battery-backed RAM and kept all other hardware/software the same. What read/write bandwidth could you expect to achieve? Explain with a one sentence answer.

Bandwidth wouldn't change because the transfer is bound by the SSI clock.

4) How many files can you store on your disk? Briefly explain how you could increase this number (do not do it, just explain how it could have been done).

512 x 512 files; to increase this number we would simply need to add indirectly-mapped data blocks to disk space.

5) Does your system allow for two threads to simultaneously stream debugging data onto one file? If yes, briefly explain how you handled the thread synchronization. If not, explain in detail how it could have been done. Do not do it, just give 4 or 5 sentences and some C code explaining how to handle the synchronization.

Multiple file accessors is basically the readers-writer problem. We would need to implement a readers and writer semaphore for each file that is opened from the disk.

eg.

void ropen(struct inode \*inod) {

OS\_bWait(&inod->readers);

++inod->readers;

if(inod->readers == 1)

OS\_bWait(&inod->writer);

OS\_bSignal(&inod->readers);

}

void rclos(struct inode \*inod) {

OS\_bWait(&inod->readers);

--inod->readers;

if(inod->readers == 0)

OS\_bSignal(&inod->writer);

OS\_bSignal(&inod->readers);

}

void wopen(struct inode \*inod) {

OS\_bWait(&inod->writer);

}

void wclos(struct inode \*inod) {

OS\_bSignal(&inod->writer);

}